AiLab - Connect-4

Colafranceschi Luca 1963232
Ippoliti Beatrice 1982749

1 Introduction

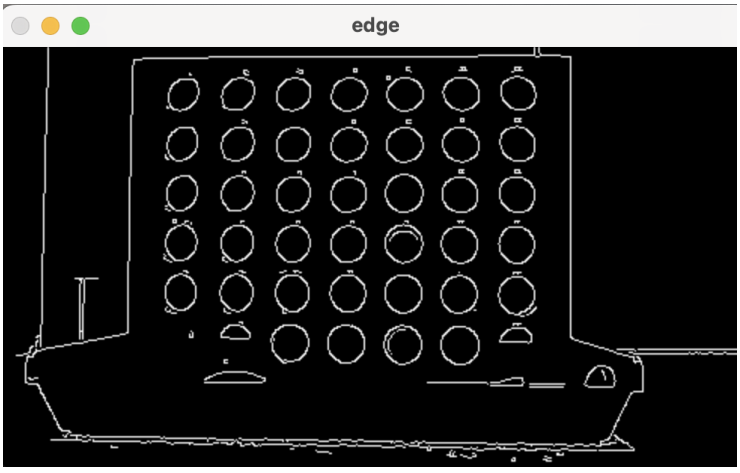
The purpose of the project is to create a program that, through the camera's frame, recognizes the *Connect-4* game board and, based on the tokens present on the board, suggests the best move to make.

To create and optimize the code, various libraries such as *OpenCV* and *PyTorch* are used, and to support the phone's camera through the program, an external application has been installed.

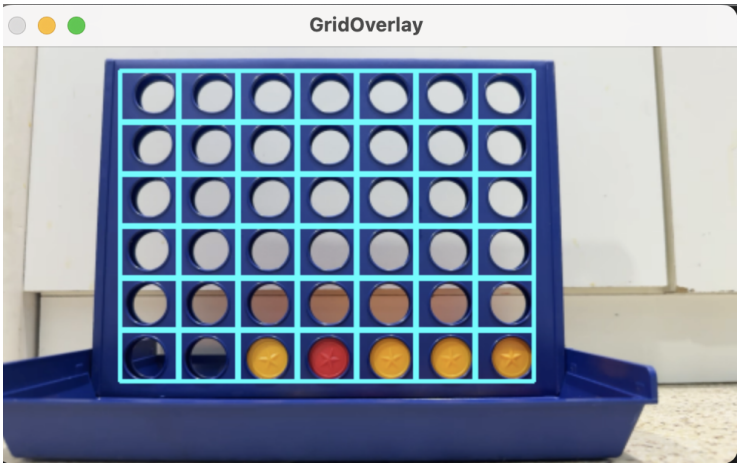
2 Method

Thanks to the use of the *OpenCV* library and the external application, we can use our phone's camera to capture the game board in order to identify the various elements within it.

Filters are applied to the frame captured by the camera to reduce noise (*bilateralFilter*) and to detect edges (*Canny*).



Subsequently, only the contours with a number of vertices resembling a circle are considered. This is necessary to delimit the game board, as by considering only the outermost circles found, we can calculate the approximate size of the board.



After delimiting the game board, it is then divided into cells, and each of these cells is passed to the neural network.

Our *Convolutional Neural Network*, implemented with *PyTorch*, is previously trained using a dataset we created, containing various photos.

For the creation of the dataset, we took multiple photos for each token, with different angles and lighting intensities, to increase the model's accuracy. For the empty cells, we varied the backgrounds to improve the recognition of the Convolutional Neural Network in all circumstances. Each photo is rotated 90° four times to increase the dataset size and ensure better accuracy. Regarding the CNN structure, we created the *OurCNN* class with several layers:

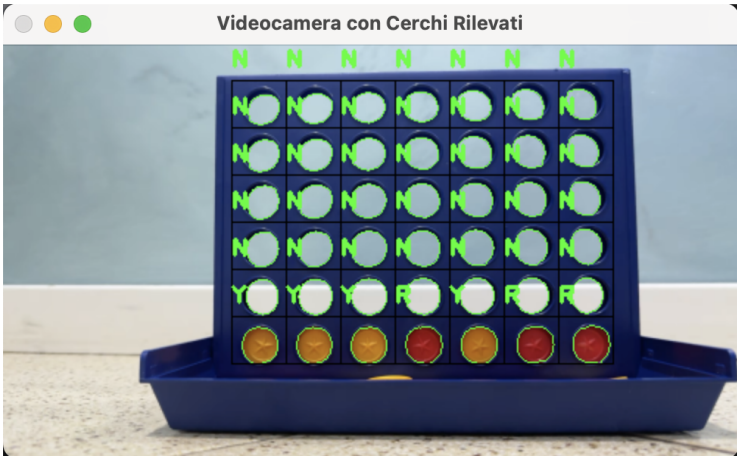
- **Conv1**: It takes an image with three channels (RGB) as input and produces 12 output channels to capture the various features of the image.
- **BatchNorm1**: This batch layer is used to normalize the output of the previous layer.
- **ReLu**: This function applies the non-linear transformation to the outputs of the network layers.
- **MaxPool**: This layer reduces the width and height dimensions of the inputs, halving them.

- **Conv2**: This is a second convolutional layer that takes the 12 output channels from the previous layer as input and produces 20 output channels.
- **Conv3**: Third convolutional layer that takes the 20 output channels from the previous layer as input and produces 32 output channels.
- **BatchNorm2**: This is a second batch layer that is used to normalize the output of the Conv3 layer.
- **fc**: This is the fully connected layer, which takes a vector as input and produces the output with the classes, in this case, 3.

The implemented *forward* function passes the outputs of each layer to the next layer until the final output is obtained.

The images in the dataset intended for training, in addition to the previously described operations, are resized (150x150 pixels), then randomly rotated up to a maximum of 10 degrees, and after being converted into PyTorch tensors, they are normalized.

Thanks to the CNN, each cell is analyzed by passing the portion of the frame, resized to the input expected by the neural network, to determine if the cell under consideration is empty ("N"), contains the red token ("R"), or the yellow token ("Y").



The results obtained are translated into a matrix that will be the input for our function, which calculates the best next move for each player.

The function is the implementation of the *Minimax* algorithm, commonly used for games like chess, tic-tac-toe, checkers, and most two-player games.

The algorithm operates recursively to find the best possible move and evaluate the game state by scanning each column and discarding the full ones where a move cannot be made.

3 Results

During the development phase of the program, the main issue encountered was during the neural network training phase. Since we couldn't find an available dataset online that represented the various useful cases, we faced difficulties in recognizing some pieces. To address this problem, we increased the number of photos in the dataset and experimented with different training configurations by modifying the number of epochs, the learning rate, and the batch size.

Once the code is run, after positioning the game board and the camera, the program scans the board and suggests the best move by drawing a square around the cell, in the same color as the piece that should be placed in that spot.

After placing the piece in the indicated cell, the best move for the other player is immediately suggested.

When one of the players achieves victory, a message is displayed indicating the winner of the game.